US006125391A

United States Patent [19]

Meltzer et al.

[11] **Patent Number: 6,125,391**

[45] **Date of Patent: *Sep. 26, 2000**

[54] **MARKET MAKERS USING DOCUMENTS FOR COMMERCE IN TRADING PARTNER NETWORKS**

[75] Inventors: **Bart Alan Meltzer**, Aptos; **Terry Allen**, Sebastopol; **Matthew Daniel Fuchs**, Los Gatos; **Robert John Glushko**, San Francisco, all of Calif.; **Murray Maloney**, Pickering, Canada

[73] Assignee: **Commerce One, Inc.**, Mountain View, Calif.

[*] Notice: This patent is subject to a terminal disclaimer.

[21] Appl. No.: **09/173,854**

[22] Filed: **Oct. 16, 1998**

[51] **Int. Cl.**^{7} **G06F 13/00**

[52] **U.S. Cl.** **709/223**; 707/513; 705/26; 709/230; 370/466

[58] **Field of Search** 709/223, 230; 705/26; 707/513; 370/466

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,742,845	4/1998	Wagner	710/11
6,012,098	1/2000	Bayeh et al.	709/246

FOREIGN PATENT DOCUMENTS

0 704 795 A1	of 1996	European Pat. Off.	G06F 9/44

OTHER PUBLICATIONS

"W3C: Extensible Markup Language (XML) 1.0—W3C Recommendation Feb. 10, 1998", http://www.w3.org/TR/1998/REC-xml-19980210, Printed from Internet Feb. 17, 1998, pp. 1–37.

Kimbrough, et al., "On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech Act Theory and Expressive Felicity", ACM Transactions on Information Systems, vol. 15, No. 4, Oct. 1997, pp. 321–367.

Fuchs, Matthew, "Domain Specific Languages for ad hoc Distributed Applications", USENIX Associate, Conference on Domain-Specific Languages, Oct. 15–17, 1997, pp. 27–35.

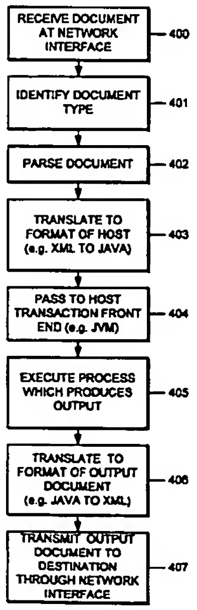
Finin, et al., "KQML as an Agent Communication Language", Association of Computing Machinery, 1994, pp. 456–463.

Primary Examiner—Kenneth R. Coulter
Attorney, Agent, or Firm—Wilson, Sonsini, Goodrich & Rosati

[57] **ABSTRACT**

A market making node in a network routes machine readable documents to connect businesses with customers, suppliers and trading partners. The self defining electronic documents, such as XML based documents, can be easily understood amongst the partners. Definitions of these electronic business documents, called business interface definitions, are posted on the Internet, or otherwise communicated to members of the network. The business interface definitions tell potential trading partners the services the company offers and the documents to use when communicating with such services. Thus, a typical business interface definition allows a customer to place an order by submitting a purchase order or a supplier checks availability by downloading an inventory status report. Also, the registration at a market maker node of a specification of the input and output documents, coupled with interpretation information in a common business library, enables participants in a trading partner network to execute the transaction in a way which closely parallels the way in which paper based businesses operate.

47 Claims, 16 Drawing Sheets


RECEIVE DOCUMENT AT NETWORK INTERFACE
400
IDENTIFY DOCUMENT TYPE
401
PARSE DOCUMENT
402
TRANSLATE TO FORMAT OF HOST (e.g. XML TO JAVA)
403
PASS TO HOST TRANSACTION FRONT END (e.g. JVM)
404
EXECUTE PROCESS WHICH PRODUCES OUTPUT
405
TRANSLATE TO FORMAT OF OUTPUT DOCUMENT (e.g. JAVA TO XML)
406
TRANSMIT OUTPUT DOCUMENT TO DESTINATION THROUGH NETWORK INTERFACE
407

US-PAT-NO: 6125391

DOCUMENT-IDENTIFIER: US 6125391 A

TITLE: Market makers using documents for commerce in trading partner networks

---------- KWIC ---------

Abstract Text - ABTX (1):
A market making node in a network routes machine readable documents to connect businesses with customers, suppliers and trading partners. The self defining electronic documents, such as XML based documents, can be easily understood amongst the partners. Definitions of these electronic business documents, called business interface definitions, are posted on the **Internet,** or otherwise communicated to members of the network. The business interface definitions tell potential trading partners the services the company offers and the documents to use when communicating with such services. Thus, a typical business interface definition allows a customer to place an order by submitting a **purchase order** or a supplier checks availability by downloading an inventory status report. Also, the registration at a market maker node of a specification of the input and output documents, coupled with interpretation information in a common business library, enables participants in a trading partner network to execute the transaction in a way which closely parallels the way in which paper based businesses operate.

Application Filing Date - AD (1):
19981016

Brief Summary Text - BSTX (7):
The **Internet** and other communications networks provide avenues for communication among people and computer platforms which are being used for a wide variety of transactions, including commercial transactions in which participants buy and sell goods and services. Many efforts are underway to facilitate commercial transactions on the **Internet**. However, with many competing standards, in order to execute a transaction, the parties to the transaction must agree in advance on the protocols to be utilized, and often require custom integration of the platform architectures to support such transactions. Commercial processes internal to a particular node not compatible with agreed upon standards, may require substantial rework for integration with other nodes. Furthermore, as a company commits to one standard or the other, the company becomes locked-in to a given standardized group of transacting parties, to the exclusion of others.

Brief Summary Text - BSTX (8):
A good overview of the challenges met by **Internet** commerce development is provided in Tenenbaum, et al., "Eco System: An **Internet** Commerce Architecture",

Computer, May 1997, pp. 48-55.

Brief Summary Text - BSTX (9):
To open commercial transactions on the **Internet,** standardization of architectural frameworks is desired. Platforms developed to support such commercial frameworks include IBM Commerce Point, Microsoft **Internet** Commerce Framework, Netscape ONE (Open Network Environment), Oracle NCA (Network Computing Architecture), and Sun/JAVASoft JECF (JAVA Electronic Commerce Framework).

Brief Summary Text - BSTX (10):
In addition to these proprietary frameworks, programming techniques, such as common distributed object model based on CORBA IIOP (Common Object Request Broker Architecture **Internet** ORB Protocol), are being pursued. Use of the common distributed object model is intended to simplify the migration of enterprise systems to systems which can inter-operate at the business application level for electronic commerce. However, a consumer or business using one framework is unable to execute transactions on a different framework. This limits the growth of electronic commerce systems.

Brief Summary Text - BSTX (15):
The present invention offers an infrastructure for connecting businesses with customers, suppliers and trading partners. Under the infrastructure of the present invention, companies exchange information and services using self-defining, machine-readable documents, such as XML (Extensible Markup Language) based documents, that can be easily understood amongst the partners. Documents which describe the documents to be exchanged, called business interface definitions BIDs herein, are posted on the **Internet,** or otherwise communicated to members of the network. The business interface definitions tell potential trading partners the services the company offers and the documents to use when communicating with such services. Thus, a typical business interface definition allows a customer to place an order by submitting a **purchase order,** compliant with a document definition published in the BID of a party to receive the **purchase order**. A supplier is allowed to check availability by downloading an inventory status report compliant with a document definition published in the BID of a business system managing inventory data. Use of predefined, machine-readable business documents provides a more intuitive and flexible way to access enterprise applications.

Brief Summary Text - BSTX (26):
According to another aspect of the invention, the specification of the input and output documents includes interpretation information for at least one of the sets of storage units identified by the logical structure. The interpretation information in one example encodes definitions for sets of parsed characters. In another example, the interpretation information provides for content model specifications, such as requiring a specific logic structure to carry a member of a list of codes mapped to product descriptions in a **catalog**. In some systems, the storage units in a logic structure of a document may include sets of unparsed data, as suits the needs of a particular

implementation.

Brief Summary Text - BSTX (41):
The whole process of building business interface definitions and enabling servers to manage commerce according to such descriptions is facilitated by a common business library, or repository, of information models for generic business concepts including business description primitives like companies, services and products, business forms like **catalogs, purchase orders** and invoices, and standard measurements, including time and date, location and classification of goods.

Detailed Description Text - DETX (2):
A detailed description of the present invention is provided with respect to the figures, in which FIG. 1 illustrates a network of market participants and market makers based on the use of business interface definitions, and supporting the trading of input and output documents specified according to such interface descriptions. The network includes a plurality of nodes 11-18 which are interconnected through a communication network such as the **Internet** 19, or other telecommunications or data communications network. Each of the nodes 11-19 consists of a computer, such as a portable computer, a desktop personal computer, a workstation, a network of systems, or other data processing resources. The nodes include memory for storing the business interface definition, processors that execute transaction processes supporting commercial transactions with other nodes in the network, and computer programs which are executed by the processors in support of such services. In addition each of the nodes includes a network interface for providing for communication across the **Internet** 19, or the other communication network.

Detailed Description Text - DETX (5):
In this example, the market participant 18 is connected directly to the market maker 17, rather than through the **Internet** 19. This connection directly to the market maker illustrates that the configuration of the networks supporting commercial transactions can be very diverse, incorporating public networks such as the **Internet** 19, and private connections such as a local area network or a Point-to-Point connection as illustrated between nodes 17 and 18. Actual communication networks are quite diverse and suitable for use to establish commercial transaction networks according to the present invention.

Detailed Description Text - DETX (8):
Thus in an exemplary system, participant nodes in the network establish virtual enterprises by interconnecting business systems and services with XML encoded documents that businesses accept and generate. For example, the business interface definition of a particular service establishes that if a document matching the BID of a request for a **catalog** entry is received, then a document matching a BID of a **catalog** entry will be returned. Also, if a document matching the BID of a **purchase order** is received, and it is acceptable to the receiving terminal, a document matching the BID of an invoice will be returned. The nodes in the network process the XML documents before they enter the local business system, which is established according to the variant

transaction processing architecture of any given system in the network. Thus, the system unpacks sets of related documents, such as MIME-encoded sets of XML documents, parses them to create a stream of "mark-up messages". The messages are routed to the appropriate applications and services using for example an event listener model like that described below.

Detailed Description Text - DETX (9):
The documents exchanged between business services are encoded using an XML language built from a repository of building blocks (a common business language) from which more complex document definitions may be created. The repository stores modules of interpretation information that are focused on the functions and information common to business domains, including business description primitives like companies, services and products; business forms like **catalogs, purchase orders** and invoices; standard measurements, like time, date, location; classification codes and the like providing interpretation information for logical structures in the XML documents.

Detailed Description Text - DETX (13):
The service type element above illustrates interpretation information carried by a business interface definition, in this example a content form allowing identification of any one of a list of valid service types. Other interpretation information includes data typing, such as for example the element &lt;H3&gt; **Internet** Address &lt;/H3&gt; including the content form "url" and expressed in the data type "string." Yet other interpretation information includes mapping of codes to elements of a list, such as for example the element &lt;H3&gt; State &lt;/H3&gt; including the code mapping for states in the file "COUNTRY.US.SUBENTITY."

Detailed Description Text - DETX (21):
The parser 301 takes a **purchase order** like that in the example above, or other document, specified according to the business interface definition and creates a set of events that are recognized by the local transaction processing architecture, such as a set of JAVA events for a JAVA virtual machine.

Detailed Description Text - DETX (23):
For example, the **purchase order** specified above may be monitored by programs listening for events generated by the parser, which would connect the document or its contents to an order entry program. Receipt of product descriptions within the **purchase order,** might invoke a program to check inventory. Receipt of address information within the **purchase order,** would then invoke a program to check availability of services for delivery. Buyer information fields in the document, could invoke processes to check order history for credit worthiness or to offer a promotion or similar processing based on knowing the identity of the consumer.

Detailed Description Text - DETX (24):
Complex listeners can be created as configurations of primitive ones. For example, a **purchase order** listener may contain and invoke the list listeners set out in the previous paragraph, or the list members may be invoked on their

own. Note that the applications that the listeners run are unlikely to be native XML processes or native JAVA processes. In these cases, the objects would be transformed into the format required by the receiving trans application. When the application finishes processing, its output is then transformed back to the XML format for communication to other nodes in the network.

Detailed Description Text - DETX (39):
&lt;**PURCHASES** HTML="OL"&gt;&lt;ITEM HTML="LI"&gt;&lt;NAME HTML="B"&gt;STUFF&lt;/NAME&gt;&lt;PRICE HTML="B"&gt;123&lt;/PRICE&gt;&lt;/ITEM&gt;&lt;/PURCHASES&gt;

Detailed Description Text - DETX (52):
FIG. 8 is a heuristic diagram showing logical structures stored in the repository in the business interface definition builder 700. Thus, the repository storage representative party business interface definitions 800, including for example a consumer BID 801, a **catalog** house BID 802, a warehouse BID 803, and an auction house BID 804. Thus, a new participant in an online market may select as a basic interface description one of the standardized BIDs which best matches its business. In addition, the repository will store a set of service business interface definitions 805. For example, an order entry BID 806, an order tracking BID 807, an order fulfillment BID 808, and a **catalog** service BID 809 could be stored. As a new participant in the market builds a business interface definition, it may select the business interface definitions of standardized services stored in the repository.

Detailed Description Text - DETX (53):
In addition to the party and service BIDs, input and output document BIDs are stored in the repository as indicated by the field 810. Thus, a **purchase order** BID 811, an invoice BID 812, a request for quote BID 813, a product availability report BID 814, and an order status BID 815 might be stored in the repository.

Detailed Description Text - DETX (59):
This XML fragment defines a service consisting of two transactions, one for taking orders and the other for tracking them. Each definition expresses a contract or promise to carry out a service if a valid request is submitted to the specified Web address. The Order service here requires an input document that conforms to a standard "po.dtd" Document Type Definition located in the repository, which may be local, or stored in an industry wide registry on the network. If a node can fulfill the order, it will return a document conforming to a customized "invoice.dtd" whose definition is local. In effect, the company is promising to do business with anyone who can submit a **Purchase Order** that conforms to the XML specification it declares. No prior arrangement is necessary.

Detailed Description Text - DETX (60):
The DTD is the formal specification or grammar for documents of a given type; it describes the elements, their attributes, and the order in which they

must appear. For example, **purchase orders** typically contain the names and addresses of the buyer and seller, a set of product descriptions, and associated **terms and conditions** such as price and delivery dates. In Electronic Data Interchange EDI for example, the X12 850 specification is a commonly used model for **purchase orders**.

Detailed Description Text - DETX (64):
business forms like **catalogs, purchase orders,** and invoices;

Detailed Description Text - DETX (66):
This information is represented as an extensible, public set of XML building blocks that companies can customize and assemble to develop XML applications quickly. Atomic CBL elements implement industry messaging standards and conventions such as standard ISO codes for countries, currencies, addresses, and time. Low level CBL semantics have also come from analysis of proposed metadata frameworks for **Internet** resources, such as Dublin Core.

Detailed Description Text - DETX (67):
The next level of elements use these building blocks to implement the basic business forms such as those used in X12 EDI transactions as well as those used in emerging **Internet** standards such as OTP (Open Trading Protocol) and OBI (Open Buying on the **Internet**).

Detailed Description Text - DETX (68):
CBL's focus is on the functions and information that are common to all business domains (business description primitives like companies, services, and products; business forms like **catalogs, purchase orders,** and invoices; standard measurements, date and time, location, classification codes). CBL builds on standards or industry conventions for semantics where possible (e.g., the rules that specify "day/month/year" in Europe vs "month/day/year" in the U.S. are encoded in separate CBL modules).

Detailed Description Text - DETX (75):
In FIG. 10, an Airbill 1000 is being defined by customizing a generic **purchase order** DTD 1001, adding more specific information about shipping weight 1002. The generic **purchase order** 1001 was initially assembled from the ground up out of CBL modules for address, date and time, currency, and vendor and product description. Using CBL thus significantly speeds the development and implementation of XML commerce applications. More importantly, CBL makes it easier for commercial applications to be interconnected.

Detailed Description Text - DETX (82):
The role of CBL and the BID processor of the present invention in an XML/JAVA environment can be further understood by the following explanation of the processing of a **Purchase Order**.

Detailed Description Text - DETX (83):
Company A defines its **Purchase Order** document type using a visual programming environment that contains a library of CBL DTDs and modules, all

defined using common business language elements so that they contain data type and other interpretation information. Company A's PO might just involve minor customizations to a more generic "transaction document" specification that comes with the CBL library, or it might be built from the ground up from CBL modules for address, date and time, currency, etc.

Detailed Description Text - DETX (85):
A compiler takes the **purchase order** definition and generates several different target forms. All of these target forms can be derived through "tree to tree" transformations of the original specification. The most important for this example are:

Detailed Description Text - DETX (86):
(a) the XML DTD for the **purchase order**.

Detailed Description Text - DETX (87):
(b) a JAVA Bean that encapsulates the data structures for a **purchase order** (the JAVA classes, arguments, datatypes, methods, and exception structures are created that correspond to information in the Schema definition of the **purchase order**).

Detailed Description Text - DETX (88):
(c) A "marshaling" program that converts **purchase orders** that conform to the **Purchase Order** DTD into a **Purchase Order** JAVA Bean or loads them into a database, or creates HTML (or an XSL style sheet) for displaying **purchase orders** in a browser.

Detailed Description Text - DETX (89):
(d) An "unmarshaling" program that extracts the data values from **Purchase Order** JAVA Beans and converts them into an XML document that conforms to the **Purchase Order** DTD.

Detailed Description Text - DETX (90):
Now, back to the scenario. A purchasing application generates a **Purchase Order** that conforms to the DTD specified as the service interface for a supplier who accepts **purchase orders**.

Detailed Description Text - DETX (91):
The parser uses the **purchase order** DTD to decompose the **purchase order** instance into a stream of information about the elements and attribute values it contains. These "property sets" are then transformed into corresponding JAVA event objects by wrapping them with JAVA code. This transformation in effect treats the pieces of marked-up XML document as instructions in a custom programming language whose grammar is defined by the DTD. These JAVA events can now be processed by the marshaling applications generated by the compiler to "load" JAVA Bean data structures.

Detailed Description Text - DETX (95):
For the example **purchase order** here, there might be listeners for:

Detailed Description Text - DETX (96):
the **purchase order,** which would connect it to an order entry program,

Detailed Description Text - DETX (100):
Complex listeners can be created as configurations of primitive ones (e.g., a **purchase order** listener may contain and invoke these listeners here, or they may be invoked on their own).

Detailed Description Text - DETX (103):
The market maker is a server that binds together a set of internal and external business services to create a virtual enterprise or trading community. The server parses incoming documents and invokes the appropriate services by, for example, handing off a request for product data to a **catalog** server or forwarding a **purchase order** to an ERP system. The server also handles translation tasks, mapping the information from a company's XML documents onto document formats used by trading partners and into data formats required by its legacy systems.

Detailed Description Text - DETX (104):
With respect to the service definition above, when a company submits a **purchase order,** the XML parser in the server uses the **purchase order** DTD to transform the **purchase order** instance into a stream of information events. These events are then routed to any application that is programmed to handle events of a given type; in some cases, the information is forwarded over the **Internet** to a different business entirely. In the **purchase order** example, several applications may act on information coming from the parser:

Detailed Description Text - DETX (105):
An order entry program processes the **purchase order** as a complete message;

Detailed Description Text - DETX (106):
An ERP system checks inventory for the products described in the **purchase order**;

Detailed Description Text - DETX (114):
FIG. 15 illustrates the processor, components and sequence of processing of incoming data at market maker node according to the present invention. The market maker node includes a communication agent 1500 at the network interface. The communication agent is coupled with an XML parser 1501 which supplies events to an XML processor 1502. The XML processor supplies events to a document router. The document router feeds a document service 1504 that provides an interface for supplying the received documents to the enterprise solution software 1505 in the host system. The communication agent 1500 is an **Internet** interface which includes appropriate protocol stacks supporting such protocols as HTTP, SMTP, FTP, or other protocols. Thus, the incoming data could come in an XML syntax, an ASCII data syntax or other syntax as suits a particular communication channel. All the documents received in non-XML syntaxes are translated into XML and passed the XML parser. A translation

table 1506 is used to support the translation from non-XML form into XML form.

Detailed Description Text - DETX (120):
"if you send me a request for a **catalog**, I will send you a **catalog**:

Detailed Description Text - DETX (121):
"if you send me a **purchase order** and I can accept it, I will send you an invoice".

Detailed Description Paragraph Table - DETL (2):
&lt;DTD NAME="markpart.dtd"&gt; &lt;H2&gt;Market Participant&lt;/H2&gt; &lt;H3&gt;Market Participant&lt;/H3&gt; &lt;ELEMENTTYPE NAME="market.participant"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;A Market Participant</TITLE&gt; &lt;SYNOPSIS&gt;A business or person and its service interfaces.&lt;/SYNOPSIS&gt; &lt;P&gt;A market participant is a document definition that is created to describe a business and at least one person with an email address and it presents a set of pointers to service interfaces located on the network. In this example the pointers have been resolved and the complete BID is presented here.&lt;/P&gt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;CHOICE&gt; &lt;ELEMENT NAME="business"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT NAME="person"&gt;&lt;/ELEMENT&gt; &lt;/CHOICE&gt;&lt;/MODEL&gt;&lt;/ELEMENTTYPE&gt; &lt;H3&gt;Party Prototype&lt;/H3&gt; &lt;PROTOTYPE NAME="party"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;The Party Prototype&lt;/TITLE&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;SEQUENCE&gt; &lt;ELEMENT NAME="party.name"OCCURS="+"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT NAME="address.set"&gt;&lt;/ELEMENT&gt; &lt;/SEQUENCE&gt;&lt;/MODEL&gt; &lt;/PROTOTYPE&gt; &lt;H3&gt;Party Types&lt;/H3&gt; &lt;ELEMENTTYPE NAME="business"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;A Business&lt;/TITLE&gt; &lt;SYNOPSIS&gt;A business (party) with a business number attribute.&lt;/SYNOPSIS&gt; &lt;P&gt;This element inherits the content model of the party prototype and adds a business number attribute, which serves as a key for database lookup. The business number may be used as a cross-reference to/from customer id, credit limits contacts lists, etc.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;EXTENDS HREF="party"&gt; &lt;ATTDEF NAME="business.number"&gt;&lt;/REQUIRED&gt;&lt;/REQUIRED&gt;&lt;/ATTDBF&gt; &lt;/EXTENDS&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Person Name&lt;/H3&gt; &lt;ELEMENTTYPE NAME="person"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;A Person&lt;/TITLE&gt;/EXPLAIN&gt; &lt;EXTENDS HREF="party"&gt; &lt;ATTDEF NAME="SSN"&gt;&lt;IMPLIED&gt;&lt;/IMPLIED&gt;&lt;/ATTDEF&gt; &lt;/EXTENDS&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Party Name&lt;/H3&gt; &lt;ELEMENTTYPE NAME="party.name"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;A Party's Name&lt;/TITLE&gt; &lt;SYNOPSIS&gt;A party's name in a string of character.&lt;/SYNOPSIS&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Address Set&lt;/H3&gt; &lt;ELEMENTTYPE NAME="address.set"&gt; &lt;MODEL&gt;&lt;SEQUENCE&gt; &lt;ELEMENT NAME="address.physical"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT NAME="telephone"OCCURS="*"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT

NAME="fax"OCCURS="*"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="email"OCCURS="*"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="**internet**"OCCURS="*"&gt;&lt;/ELEMENT&gt; &lt;/SEQUENCE&gt;</MODEL>
&lt;/ELEMENTTYPE&gt; &lt;H3&gt;Physical Address&lt;/H3&gt; &lt;ELEMENTTYPE
NAME="address.physical"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Physical
Address&lt;/TITLE&gt; &lt;SYNOPSIS&gt;The street address, city, state, and zip
code.&lt;/SYNOPSIS&gt;&lt;/EXPLA IN&gt; &lt;MODEL&gt;&lt;SEQUENCE&gt;
&lt;ELEMENT NAME="street"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="city"></ELEMENT&gt; &lt;ELEMENT NAME="state"></ELEMENT&gt;
&lt;ELEMENT NAME="postcode"OCCURS="?"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="country"&gt;&lt;/ELEMENT&gt; &lt;/SEQUENCE&gt;</MODEL>
&lt;/ELEMENTTYPE&gt; &lt;H3&gt;Street&lt;/H3&gt; &lt;ELEMENTTYPE
NAME="street"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Street Address&lt;/TITLE&gt;
&lt;SYNOPSIS&gt;Street or postal address.&lt;/SYNOPSIS&gt;&lt;/EXPLAIN&gt;
&lt;MODEL&gt;&lt;STRING&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;
&lt;H3&gt;City&lt;/H3&gt; &lt;ELEMENTTYPE NAME="city"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;City Name or Code&lt;/TITLE&gt; <P&gt;The city
name or code is a string that contains sufficient information to identity a
city within a designated state.&lt;/P&gt; &lt;/EXPLAIN&gt;
&lt;MODEL&gt;&lt;STRING&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;
&lt;H3&gt;State&lt;/H3&gt; &lt;ELEMENTTYPE NAME="state"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;State, Province or Prefecture Name or
Code&lt;/TITLE&gt; &lt;P&gt;The state name or code contains sufficient
information to identfy a state within a designated
country.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING
DATATYPE="COUNTRY.US.SUBENTITY"&gt;&lt;/STRING&gt;&lt;/MODEL&gt;
&lt;/ELEMENTTYPE&gt; &lt;H3&gt;Postal Code&lt;/H3&gt; &lt;ELEMENTTYPE
NAME="postcode"> <EXPLAIN&gt;&lt;TITLE&gt;Postal Code</TITLE&gt;
&lt;P&gt;A postal code is an alphanumeric code, designated by an appropriate
postal authority, that is used to identfy a location or region within the
jurisdiction of that postal authority. Postal authorities include designated
national postal authorities.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING
DATATYPE="string"&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;
&lt;H3&gt;Country&lt;/H3&gt; &lt;ELEMENTTYPE NAME="country"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;Country Code&lt;/TITLE&gt; &lt;P&gt;A country code
is a two-letter code, designated by ISO, that is used to uniquely identfy a
country.&lt;/P&gt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING
DATATYPE="country"&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;
&lt;H3&gt;Network Addresses&lt;/H3&gt; &lt;ELEMENTTYPE NAME="telephone"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;Telephone Number&lt;TITLE&gt; &lt;P&gt;A telephone
number is a string of alphanumerics and punctuation that uniquely identifies a
telephone service terminal, including extension
number.&lt;P>&lt;/EXPLAIN&gt;
&lt;MODEL&gt;&lt;STRING&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;
&lt;H3&gt;Fax&lt;/H3&gt; &lt;ELEMENTTYPE NAME="fax"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;Fax Number&lt;/TITLE&gt; &lt;P&gt;A fax number is
a string of alphanumerics and punctuation that uniquely identifies a fax
service terminal.</P&gt; &lt;/EXPLAIN&gt;
&lt;MODEL&gt;&lt;STRING&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;

&lt;H3&gt;Email&lt;/H3&gt; &lt;ELEMENTTYPE NAME="email"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;Email Address&lt;/TITLE&gt; &lt;P&gt;An email
address is a datatype-constrained string that uniquely identifies a mailbox on
a server.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING
DATATYPE="email"&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt;
<H3>**Internet** Address&lt;/H3&gt; &lt;ELEMENTTYPE NAME="**internet**">
&lt;EXPLAIN&gt;&lt;TITLE>**Internet** Address&lt;/TITLE&gt; &lt;P&gt;An
Internet address is a datatype-constrained string that uniquely identifies a
resource on the **Internet** by means of a URL.&lt;/P&gt;&lt;/EXPLAIN&gt;
&lt;MODEL&gt;&lt;STRING DATATYPE="url"&gt;&lt;/STRING&gt;&lt;/MODEL&gt;
&lt;/ELEMENTTYPE&gt; &lt;/DTD&gt; &lt;/SCHEMA&gt; Service Description Sample
&lt;!DOCTYPE schema SYSTEM "bidl.dtd"&gt; &lt;SCHEMA&gt; &lt;H1&gt;Service
Description Sample BID&lt;/H1&gt; &lt;META WHO.OWNS="Veo Systems"
WHO.COPYRIGHT="Veo Systems" WHEN.COPYRIGHT="1998" DESCRIPTION="Sample
BID"
WHO.CREATED="*" WHEN.CREATED="*" WHAT.VERSION="*" WHO.MODIFIED="*"
WHEN.MODIFIED="*" WHEN.EFFECTIVE="*" WHEN.EXPIRES="*" WHO.EFFECTIVE="*"
WHO.EXPIRES="*"&gt; &lt;/META&gt; &lt;PROLOG&gt; &lt;XMLDECL
STANDALONE="no"&gt;&lt;/XMLDECL&gt; &lt;DOCTYPE NAME="service">
&lt;SYSTEM&gt;service.dtd&lt;/SYSTEM&gt;&lt;/DOCTYPE&gt; <PROLOG>
&lt;DTD NAME="service.dtd"&gt; &lt;H2&gt;Services&lt;/H2&gt;
<H3>Includes</H3&gt; <!--
INCLUDE&gt;&lt;SYSTEM&gt;comments.bim&lt;/SYSTEM&gt;&lt;/INCLUDE-->
&lt;H3&gt;Service Set&lt;/H3&gt; &lt;ELEMENTTYPE NAME="service.set">
&lt;EXPLAIN&gt;&lt;TITLE&gt;Service Set</TITLE> &lt;SYNOPSIS&gt;A set of
services.</SYNOPSIS&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt; &lt;ELEMENT
NAME="service"OCCURS="+"&gt;&lt;/ELEMENT&gt;
&lt;/MODEL&gt;&lt;/ELEMENTTYPE&gt; &lt;H3&gt;Services Prototype</H3>
&lt;PROTOTYPE NAME="prototype.service"&gt;
&lt;EXPLAIN&gt;&lt;TITLE&gt;Service&lt;/TITLE&gt;&lt;/EXPLAIN>
&lt;MODEL&gt;&lt;SEQUENCE&gt; &lt;ELEMENT
NAME="service.name"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="service.terms"OCCURS="+"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="service.location"OCCLRS="+"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="service.operation"OCCLRS="+"&gt;&lt;/ELEMENT&gt;
&lt;SEQUENCE&gt;&lt;/MODEL&gt; &lt;!-- ATTGROUP&gt;&lt;IMPLEMENTS
HREF="common.atrrib"&gt;&lt;/IMPLEMENTS&gt;&lt;/ATTGROUP &lt;/PROTOTYPE>
&lt;H3&gt;Service&lt;/H3&gt; &lt;INTRO&gt;&lt;P&gt;A service is an addressable
network resource that provides interfaces to specific operations by way of
input and output documents.&lt;/P&gt;&lt;/INTRO&gt; &lt;ELEMENTTYPE
NAME="service"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Service&lt;/TITLE&gt;
&lt;P&gt;A service is defined in terms of its name, the location(s) at which
the service is available, and the operation(s) that the service
performs.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;SEQUENCE&gt; &lt;ELEMENT
NAME="service.name"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="service.location"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="service.operation"OCCURS="+"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT
NAME="service.terms"><ELEMENT&gt; &lt;/SEQUENCE&gt;&lt;/MODEL&gt;
&lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Name&lt;/H3&gt; &lt;ELEMENTTYPE

NAME="service.name"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Name</TITLE> <P&gt;The service name is a human-readable string that ascribes a moniker for a service. It may be employed is user interfaces and documentation, or for other purposes.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING></STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Location&lt;/H3&gt; &lt;ELEMENTTYPE NAME="service.location"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Location</TITLE&gt; &lt;SYNOPSIS&gt;A LRI of a service.&lt;/SYNOPSIS&gt; &lt;P&gt;A service location is a datatype-constrained string that locates a service on the **Internet** by means of a URI.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING DATATYPE="url"></STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Operations&lt;/H3&gt; &lt;INTRO&gt;&lt;P&gt;A service operation consists of a name, location and its interface, as identified by the type of input document that the service operation accepts and by the type of document that it will return as a result.&lt;/P&gt;&lt;/INTRO&gt; &lt;ELEMENTTYPE NAME="service.operation"> &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Operations</TITLE> <P&gt;A service operation must have a name, a location, and at least one document type as an input, with one or more possible document types returned as a result of the operation.&lt;/P&gt; &lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;SEQUENCE&gt; &lt;ELEMENT NAME="service.operation.name"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT NAME="service.operation.location"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT NAME="service.operation.input"&gt;&lt;/ELEMENT&gt; &lt;ELEMENT NAME="service.operation.output"&gt;&lt;/ELEMENT> </SEQUENCE&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Operation Name&lt;/H3&gt; &lt;ELEMENTTYPE NAME="service.operation.name"> &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Operation Name&lt;/TITLE&gt;

Detailed Description Paragraph Table - DETL (3):
&lt;P&gt;The service operation name is a human readable string that ascribes a moniker to a service operation. It may be employed in user interfaces and documentation, or for other purposes.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING></STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Operation Location&lt;/H3&gt; &lt;INTRO&gt;&lt;P&gt;The service location is a network resource. That is to say, a URI.&lt;/P&gt;&lt;/INTRO&gt; &lt;ELEMENTTYPE NAME="service.operation.location"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Operation Location"&lt;/TITLE&gt; &lt;SYNOPSIS&gt;A URI of a service operation.&lt;/SYNOPSIS&gt; &lt;P&gt;A service operation location is a datatype-constrained string that locates a service operation on the **Internet** by means of a URL.&lt;/P&gt;&lt;/EXPLAIN&gt; &lt;MODEL&gt;&lt;STRING DATATYPE="url"></STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Operation Input Document&lt;/H3&gt; &lt;INTRO&gt;&lt;P&gt;The input to a service operation is defined by its input document type. That is, the service operation is invoked when the service operation location receives an input document whose type corresponds to the document type specified by this element.&lt;/P&gt; &lt;P&gt;Rather than define the expected input and output document types in the market participant document, this example provides pointers to externally-defined BIDs This

allows reuse of the same BID as the input and/or output document type for multiple operations. In addition, it encourages parallel design and implementation.&lt;/P&gt;&lt;/INTRO&gt; &lt;ELEMENTTYPE NAME="service.operation.input"> &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Operation Input</TITLE&gt; &lt;SYNOPSIS>Identifies the type of the service operation input document.&lt;/SYNOPSIS&gt; &lt;P&gt;Service location input is a datatype-constrained string that identifies a BID on the **Internet** by means of a URI.&lt;/P> </EXPLAIN> <MODEL&gt;&lt;STRING DATATYPE="url"&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Operation Output Document Type&lt;/H3&gt; &lt;INTRO&gt;&lt;P&gt;The output of a service operation is defined by its output document type(s). That is, the service operation is expected to emit a document whose type corresponds to the document type specified by this element.&lt;/P&gt;&lt;/INTRO&gt; &lt;ELEMENTTYPE NAME="service.operation.output"> <EXPLAIN&gt;&lt;TITLE&gt;Service Operation Output&lt;/TITLE&gt; &lt;SYNOPSIS>Identifies the type of the service operation output document.&lt;/SYNOPSIS&gt; &lt;P&gt;Service location output is a datatype-constrained string that identifies a BID on the **Internet** by means of a URI.&lt;/P> </EXPLAIN> <MODEL&gt;&lt;STRING DATATYPE="url"&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;H3&gt;Service Terms&lt;/H3&gt; &lt;INTRO&gt;&lt;P&gt;This is a simple collection of string elements, describing the terms of an agreement.&lt;/P&gt;&lt;/INTRO&gt; &lt;ELEMENTTYPE NAME="service.terms"&gt; &lt;EXPLAIN&gt;&lt;TITLE&gt;Service Terms</TITLE&gt; &lt;SYNOPSIS&gt;Describes the terms of a given agreement.&lt;/SYNOPSIS> </EXPLAIN> <MODEL&gt;&lt;STRING DATATYPE="string"&gt;&lt;/STRING&gt;&lt;/MODEL&gt; &lt;/ELEMENTTYPE&gt; &lt;/DTD&gt; &lt;/SCHEMA&gt;

Detailed Description Paragraph Table - DETL (6):

Market Participant DTD <!ELEMENT business (party.name+, address.set)&gt; <!ATTLIST business business.number CDATA #REQUIRED &lt;!ELEMENT party.name (#PCDATA)&gt; &lt;!ELEMENT city (#PCDATA)&gt; &lt;!ELEMENT **internet** (#PCDATA)&gt; &lt;!ELEMENT country (#PCDATA)&gt; &lt;!ELEMENT state (#PCDATA)&gt; <!ELEMENT email (#PCDATA)&gt; <!ELEMENT address.physical (street, city, state postcode?, country)&gt; &lt;!ELEMENT telephone (#PCDATA)&gt; &lt;!ELEMENT person (party.name+, address.set)&gt; &lt;!ATTLIST person SSN CDATA #IMPLIED &gt; &lt;!ELEMENT fax (#PCDATA)&gt; &lt;!ELEMENT street (#PCDATA)&gt; <!ELEMENT address.set (address.physical, telephone*, fax*, email*, internet*)> <!ELEMENT postcode (#PCDATA)&gt; &lt;!ELEMENT market.participant (business .vertline. person)&gt; Service DTD &lt;!ELEMENT service.location (#PCDATA)&gt; &lt;!ELEMENT service.terms (#PCDATA)&gt; &lt;!ELEMENT service.operation.name (#PCDATA)&gt; &lt;!ELEMENT service.operation (service.operation.name, service.operation.loc ation, service.operation.input, service.operation. output)&gt; &lt;!ELEMENT service (service.name, service.location, service.operation+, service.terms) &gt;

&lt;!ELEMENT service.operation.input (#PCDATA)&gt; &lt;!ELEMENT service.operation.location (#PCDATA)&gt; &lt;!ELEMENT service.name (#PCDATA)&gt; &lt;!ELEMENT service.set (service+)&gt; &lt;!ELEMENT service.operation.output (#PCDATA)&gt;

Detailed Description Paragraph Table - DETL (7):

<?xml version="1.0"?&gt; &lt;!--rorder.xml Version: 1.0 --&gt; &lt;!Copyright 1998 Veo Systems, Inc. --&gt; &lt;!DOCTYPE transaction.description SYSTEM "urn:x-veosystems:dtd:cbl:transac t: 1.0:> <transaction.description transaction.type="order"&gt; &lt;meta&gt; &lt;urn?urn:x-veosystems:doc:00023 &lt;/urn&gt; &lt;thread.id party.assigned.by="reqorg"&gt;FRT876 &lt;/thread.id&gt; </meta> <issuer.pointer&gt; &lt;xll.locator urllink="reqorg.xml"&gt;Customer Pointer &lt;/xll.locator&gt; &lt;/issuer.pointer&gt; &lt;counterparty.pointer&gt; &lt;xll.locator urllink="compu.xml">**Catalog** entry owner pointer </xll.locator> </counterparty.pointer&gt; &lt;exchange.description&gt; &lt;line.item&gt; &lt;product.instance&gt; &lt;product.description.pointer> &lt;xll.locator urllink="cthink.xml">**Catalogue** Entry Pointer &lt;xll.locator&gt; &lt;/product.description.pointer&gt; &lt;product.specifics&gt; &lt;info.description.set&gt; &lt;info.description&gt; &lt;xml.descriptor&gt; &lt;doctype&gt; &lt;dtd system.id="urn:x-veosystems:dtd:cbl:gprod:1.0"/> <doctype&gt; &lt;xml.descriptor.details&gt; &lt;xll.xptr.frag&gt;DESCENDANT(ALL, os)STRING("Windows 95") &lt;/xll.xptr.frag&gt; &lt;/xll.xptr.frag&gt;DECENDANT(ALL,p.speed)STRING("200") &lt;/xll.xptr.frag&gt; &lt;/xll.xptr.frag&gt;DESCENDANT(ALL,hard.disk.capacity) STRING("4") &lt;/xll.xptr.frag&gt; &lt;/xll.xptr.frag&gt;DESCENDANT(ALL,d.size)STRING("14.1") </xll.xptr.frag&gt; &lt;/xml.descriptor.details&gt; &lt;/xml.descriptor&gt; &lt;/info.description&gt; &lt;/info.description.set> </product.specifics&gt; &lt;/quantity&gt;1 &lt;/quantity&gt; &lt;/product.instance&gt; &lt;shipment.coordinates.set> <shipment.coordinates&gt; &lt;shipment.destination&gt; &lt;address.set&gt; &lt;address.named&gt;SW-1 &lt;address.named&gt; &lt;address.physical&gt; &lt;building.sublocation>208C</building.sublocation&gt; &lt;location.in.street&gt;123 &lt;/location.in. street&gt; &lt;street&gt;Frontage Rd. &lt;/street&gt; &lt;city&gt;Beltway &lt;/city&gt; &lt;country.subentity.us countiy.subentity.us.name="MD"/> <postcode&gt;20000

Detailed Description Paragraph Table - DETL (19):

Java to XML <!.DOCTYPE tree SYSTEM "tree.dtd"&gt; &lt;tree source = "null"

pass-through = "false"&gt; &lt;before&gt; &lt;vardef name =
"attribute.def"&gt; &lt;element source = "ATTRIBUTE" class = "NAME" type = "5"
position = "-2"&gt; &lt;parse&gt; &lt;data class = "java.lang.String"
position = "-2"/&gt; &lt;/parse&gt; &lt;/element&gt; &lt;/vardef&gt;
&lt;vardef name = "pcdata.def"&gt; &lt;element source = "PCDATA" class =
"NAME" type = "4"position = "-2"&gt; &lt;parse&gt; &lt;data class = "999"
type = "6" position = "-2"/&gt; &lt;/parse&gt; &lt;/element&gt;
&lt;/vardef&gt; &lt;vardef name = "content.def"&gt; &lt;element source =
"PCDATA"&gt; &lt;parse&gt; &lt;data class = "999" type = "6" position =
"-2"/&gt; &lt;/parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name =
"ServiceSet.var"&gt; &lt;element source =
"com.veo.xdk.dev.schema.test.blib.ServiceSet" class = "service.set" type = "4"
position = "-2"&gt; &lt;parse&gt; &lt;callvar name = "Service.var"&gt;
&lt;/parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name =
"PrototypeService.var"&gt; &lt;element source =
"com.veo.xdk.dev.schema.test.blib.PrototypeService" class =
"prototype.service" type = "4" position = "-2"&gt; &lt;parse&gt; &lt;callvar
name = "pcdata.def" parms = "setSource ServiceNameToXML setGenerator
service.name"/&gt; &lt;callvar name = "pcdata.def" parms = "setSource
ServiceTermsToXML setGenerator service.terms"/&gt; &lt;callvar name =
"pcdata.def" parms = "setSource ServiceLocationToXML setGenerator
service.location"/&gt; &lt;callvar name = "ServiceOperation.var"/&gt;
&lt;/parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name =
"Service.var"&gt; &lt;element source =
"com.veo.xdk.dev.schema.test.blib.Service" class = "service" type = "8"
position = "0"&gt; &lt;parse&gt; &lt;callvar name = "pcdata.def" parms =
"setSource ServiceNameToXML setGenerator service.name"/&gt; &lt;callvar name
= "pcdata.def" parms = "setSource ServiceLocationToXML setGenerator
service.location"/&gt; &lt;callvar name = "ServiceOperation.var"&gt;
&lt;callvar name = "pcdata.def" parms = "setSource ServiceTermsToXML
setGenerator service.terms"/&gt; &lt;/parse&gt; &lt;/element&gt;
&lt;/vardef&gt; &lt;vardef name = "ServiceOperation.var"/&gt; &lt;element
source = "com.veo.xdk.dev.schema.test.blib.ServiceOperation" class =
"service.operation" type = "4" position = "-2"&gt; &lt;parse&gt; &lt;callvar
name = "pcdata.def" parms = "setSource ServiceOperationNameToXM L setGenerator
service.operation.name"/&gt; &lt;callvar name = "pcdata.def" parms =
"setSource ServiceOperationLocation ToXML setGenerator
service.operation.location"/&gt; &lt;callvar name = "pcdata.def" parms =
"setSource ServiceOperationInputToX ML setGenerator
service.operation.input"/> <callvar name = "pcdata.def" parms =
"setSource ServiceOperationOutputTo XML setGenerator
service.operation.output"/&gt; &lt;/parse&gt; &lt;/element&gt;
&lt;/vardef&gt; &lt;/before&gt; &lt;parse&gt; &lt;callvar name =
"ServiceSet.var"/&gt; &lt;callvar name = "PrototypeService.var"/&gt;
&lt;callvar name = "Service.var"/> &lt;callvar name =
"ServiceOperation.var"/&gt; &lt;/parse&gt; &lt;/tree&gt; XML to Java
&lt;!DOCTYPE tree SYSTEM "tree.dtd"&gt; &lt;tree source = "null" pass-through
= "false"&gt; &lt;before&gt; &lt;vardef name = "business.var"&gt;
&lt;element source = "business" class =

"com.veo.xdk.dev.schema.test.blib.Business" type = "7" setter = "setBusiness"&gt; &lt;before&gt; &lt;onattribute name = "business.number"&gt; &lt;actions&gt; &lt;callmeth name = "businessNumberFromXML"> <parms> <getattr name = "business.number"/> </parms> </callmeth> </actions&gt; &lt;/onattribute&gt; &lt;before&gt; &lt;parse&gt; &lt;callvar name = "party.name.var" parms = "setPosition -1"/> <callvar name = "address.set.var"/&gt; &lt;/parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name = "party.name.var"&gt; &lt;element source = "party.name" setter = "partyNameFromXML" position = "-1" class = "java.lang.String"&gt; &lt;parse&gt; &lt;data class = "java.lang.String" position = "0"/> </parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name = "city.var"&gt; &lt;element source = "city" setter = "cityFromXML" position = "-1" class = "java.lang.String"&gt; &lt;parse&gt; &lt;data class = "java.lang.String" position = "0"/> </parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name = "internet.var"&gt; &lt;element source = "**internet**" setter = "internetFromXML" position = "-1" class = "java.lang.String"&gt; &lt;parse&gt; &lt;data class = "java.lang.String" position = "0"/> </parse&gt; &lt;/element&gt; &lt;/vardef&gt; &lt;vardef name = "country.var"&gt; &lt;element source = "country" setter = "countryFromXML" position = "1" class = "java.lang.String"&gt; &lt;parse&gt; &lt;data class = "java.lang.String" position = "0"/> /<parse&gt;

Other Reference Publication - OREF (1):
"W3C: Extensible Markup Language (XML) 1.0--W3C Recommendation Feb. 10, 1998", http://www.w3.org/TR/1998/REC-xml-19980210, Printed from **Internet** Feb. 17, 1998, pp. 1-37.